1.3 Round-off Error and Computer Arithmetic

The arithmetic performed by a calculator or computer is different from the arithmetic that we use in our algebra and calculus courses. From your past experience you might expect that we always have as true statements such things as $2 + 2 = 4$, $4 \cdot 8 = 32$, and $(\sqrt{3})^2 = 3$. In standard *computational* arithmetic we expect exact results for $2 + 2 = 4$ and $4 \cdot 8 = 32$, but we will not have precisely $(\sqrt{3})^2 = 3$. To understand why this is true we must explore the world of finite-digit arithmetic.

In our traditional mathematical world we permit numbers with an infinite number of digits. The arithmetic we use in this world *defines* $\sqrt{3}$ as that unique positive number that when multiplied by itself produces the integer 3. In the computational world, however, each representable number has only a fixed and finite number of digits. This means, for example, that only rational numbers—and not even all of these—can be represented exactly. Since $\sqrt{3}$ is not rational, it is given an approximate representation within the machine, a representation whose square will not be precisely 3, although it will likely be sufficiently close to 3 to be acceptable in most situations. In most cases, then, this machine representation and arithmetic is satisfactory and passes without notice or concern, but at times problems arise because of this discrepancy.

The error that is produced when a calculator or computer is used to perform real-number calculations is called *round-off error*. It occurs because the arithmetic performed in a machine involves numbers with only a finite number of digits, with the result that calculations are performed with only approximate representations of the actual numbers. In a typical computer, only a relatively small subset of the real number system is used for the representation of all the real numbers. This subset contains only rational numbers, both positive and negative, and stores the fractional part, together with an exponential part.

In 1985, the IEEE (Institute for Electrical and Electronic Engineers) published a report called *Binary Floating Point Arithmetic Standard 754–1985*. Formats were specified for single, double, and extended precisions. These standards are generally followed by all microcomputer manufacturers using hardware that performs real-number, or *floating point*, arithmetic operations. For example, the double precision real numbers require a 64-bit (binary digit) representation.

The first bit is a sign indicator, denoted s. This is followed by an 11-bit exponent, c, and a 52-bit binary fraction, f, called the **mantissa**. The base for the exponent is 2.

The normalized form for the nonzero double precision numbers have $0 < c < 2^{11} - 1 = 2047$. Since c is positive, a bias of 1023 is subtracted from c to give an actual exponent in the interval $(-1023, 1024)$. This permits adequate representation of numbers with both large and small magnitude.The first bit of the fractional part of a number is assumed to be 1 and is not stored in order to give one additional bit of precision to the representation, Since 53 binary digits correspond to between 15 and 16 decimal digits, we can assume that a number represented using this system has at least 15 decimal digits of precision. Thus, numbers represented in normalized double precision have the form

$$(-1)^s * 2^{c-1023} * (1+f).$$

Consider for example, the machine number

0 10000000011 1011100100010000000000000000000000000000000000000000.

The leftmost bit is zero, which indicates that the number is positive. The next 11 bits, 10000000011, giving the exponent, are equivalent to the decimal number

$$c = 1 \cdot 2^{10} + 0 \cdot 2^9 + \cdots + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 1024 + 2 + 1 = 1027.$$

The exponential part of the number is, therefore, $2^{1027-1023} = 2^4$. The final 52 bits specify that the mantissa is

$$f = 1 \cdot \left(\frac{1}{2}\right)^1 + 1 \cdot \left(\frac{1}{2}\right)^3 + 1 \cdot \left(\frac{1}{2}\right)^4 + 1 \cdot \left(\frac{1}{2}\right)^5 + 1 \cdot \left(\frac{1}{2}\right)^8 + 1 \cdot \left(\frac{1}{2}\right)^{12}.$$

As a consequence, this machine number precisely represents the decimal number

$$\begin{aligned}
&(-1)^s * 2^{c-1023} * (1+f) \\
&\quad = (-1)^0 \cdot 2^{1027-1023} \left(1 + \left(\frac{1}{2} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{256} + \frac{1}{4096}\right)\right) \\
&\quad = 27.56640625.
\end{aligned}$$

However, the next smallest machine number is

0 10000000011 1011100100001111111111111111111111111111111111111111

and the next largest machine number is

0 10000000011 1011100100010000000000000000000000000000000000000001.

This means that our original machine number represents not only 27.56640625, but also half of all the real numbers that are between 27.56640625 and its two nearest machine-number neighbors. To be precise, it represents any real number in the interval

$$[27.56640624999999988897769753748434595763683319091796875,$$
$$27.56640625000000011102230246251565404236316680908203125).$$

The smallest normalized positive number that can be represented has $s = 0$, $c = 1$, and $f = 0$, and is equivalent to the decimal number

$$2^{-1022} \cdot (1+0) \approx 0.225 \times 10^{-307},$$

The largest normalized positive number that can be represented has $s = 0$, $c = 2046$, and $f = 1 - 2^{-52}$, and is equivalent to the decimal number

$$2^{1023} \cdot \left(1 + \left(1 - 2^{-52}\right)\right) \approx 0.17977 \times 10^{309}.$$

Numbers occurring in calculations that have too small a magnitude to be represented result in **underflow**, and are generally set to 0 with computations continuing. However, numbers occurring in calculations that have too large a magnitude to be represented result in **overflow** and typically cause the computations to stop (unless the program has been designed to detect this occurrence). Note that there are two representations for the number zero; a positive 0 when $s = 0$, $c = 0$ and $f = 0$ and a negative 0 when $s = 1$, $c = 0$ and $f = 0$. The use of binary digits tends to complicate the computational problems that occur when a finite collection of machine numbers is used to represent all the real numbers. To examine these problems, we now assume, for simplicity, that machine numbers are represented in the normalized *decimal* form

$$\pm 0.d_1 d_2 \dots d_k \times 10^n, \quad 1 \le d_1 \le 9, \quad 0 \le d_i \le 9$$

for each $i = 2, \dots, k$. Numbers of this form are called *k-digit decimal machine numbers*.

Any positive real number within numerical range of the machine can be normalized to achieve the form

$$y = 0.d_1 d_2 \dots d_k d_{k+1} d_{k+2} \dots \times 10^n.$$

The **floating-point form** of y, denoted by $fl(y)$, is obtained by terminating the mantissa of y at k decimal digits. There are two ways of performing the termination. One method, called **chopping**, is to simply chop off the digits $d_{k+1}d_{k+2}\dots$ to obtain

$$fl(y) = 0.d_1 d_2 \dots d_k \times 10^n.$$

The other method of terminating the mantissa of y at k decimal points is called **rounding**. If the $k+1$st digit is smaller than 5, then the result is the same as chopping. If the $k+1$st digit is 5 or greater, then 1 is added to the kth digit and the resulting number is chopped. As a consequence, rounding can be accomplished by simply adding $5 \times 10^{n-(k+1)}$ to y and then chopping the result to obtain $fl(y)$. Note that when rounding up the exponent n could increase by 1. In summary, when rounding we add one to d_k to obtain $fl(y)$ whenever $d_{k+1} \ge 5$, that is, we round up; when $d_{k+1} < 5$, we chop off all but the first k digits, so we round down.

The next examples illustrate floating-point arithmetic when the number of digits being retained is quite small. Although the floating-point arithmetic that is performed on a calculator or computer will retain many more digits, the problems this arithmetic can cause are essentially the same regardless of the number of digits. Retaining more digits simply postpones the awareness of the situation.

EXAMPLE 1 The irrational number π has an infinite decimal expansion of the form $\pi = 3.14159265\dots$. Written in normalized decimal form, we have

$$\pi = 0.314159265\dots \times 10^1.$$

The five-digit floating-point form of π using chopping is

$$fl(\pi) = 0.31415 \times 10^1 = 3.1415.$$

Since the sixth digit of the decimal expansion of π is a 9, the five-digit floating-point form of π using rounding is

$$fl(\pi) = (0.31415 + 0.00001) \times 10^1 = 0.31416 \times 10^1 = 3.1416. \quad \square$$

The error that results from replacing a number with its floating-point form is called **round-off error** (regardless of whether the rounding or chopping method is used). There are two common methods for measuring approximation errors.

The approximation p^* to p has **absolute error** $|p - p^*|$ and **relative error** $|p - p^*|/|p|$, provided that $p \neq 0$.

EXAMPLE 2

a. If $p = 0.3000 \times 10^1$ and $p^* = 0.3100 \times 10^1$, the absolute error is 0.1 and the relative error is $0.333\overline{3} \times 10^{-1}$.

b. If $p = 0.3000 \times 10^{-3}$ and $p^* = 0.3100 \times 10^{-3}$, the absolute error is 0.1×10^{-4}, but the relative error is again $0.333\overline{3} \times 10^{-1}$.

c. If $p = 0.3000 \times 10^4$ and $p^* = 0.3100 \times 10^4$, the absolute error is 0.1×10^3, but the relative error is still $0.333\overline{3} \times 10^{-1}$.

This example shows that the same relative error can occur for widely varying absolute errors. As a measure of accuracy, the absolute error can be misleading and the relative error is more meaningful, since the relative error takes into consideration the size of the true value. $\square$

The arithmetic operations of addition, subtraction, multiplication, and division performed by a computer on floating-point numbers also introduce error. These arithmetic operations involve manipulating binary digits by various shifting and logical operations, but the actual mechanics of the arithmetic are not pertinent to our discussion. To illustrate the problems that can occur, we simulate this *finite-digit arithmetic* by first performing, at each stage in a calculation, the appropriate operation using exact arithmetic on the floating-point representations of the numbers. We then convert the result to decimal machine-number representation. The most common round-off error producing arithmetic operation involves the subtraction of nearly equal numbers.

EXAMPLE 3

Suppose we use four-digit decimal chopping arithmetic to simulate the problem of performing the computer operation $\pi - \frac{22}{7}$. The floating-point representations of these numbers are

$$fl(\pi) = 0.3141 \times 10^1 \qquad \text{and} \qquad fl\left(\frac{22}{7}\right) = 0.3142 \times 10^1.$$

Performing the exact arithmetic on the floating-point numbers gives

$$fl(\pi) - fl\left(\frac{22}{7}\right) = -0.0001 \times 10^1,$$

which converts to the floating-point approximation of this calculation:

$$p^* = fl\left(fl(\pi) - fl\left(\frac{22}{7}\right)\right) = -0.1000 \times 10^{-2}.$$

Although the relative errors using the floating-point representations for π and $\frac{22}{7}$ are small,

$$\left|\frac{\pi - fl(\pi)}{\pi}\right| \leq 0.0002 \quad \text{and} \quad \left|\frac{\frac{22}{7} - fl\left(\frac{22}{7}\right)}{\frac{22}{7}}\right| \leq 0.0003,$$

the relative error produced by subtracting the nearly equal numbers π and $\frac{22}{7}$ is about 700 times as large:

$$\left|\frac{\left(\pi - \frac{22}{7}\right) - p^*}{\left(\pi - \frac{22}{7}\right)}\right| \approx 0.2092. \quad \square$$

Rounding arithmetic is easily implemented in Maple. The statement

```
>Digits:=t;
```

causes all arithmetic to be rounded to t digits. For example, $fl(fl(x) + fl(y))$ is performed using t-digit rounding arithmetic by

```
>evalf(evalf(x)+evalf(y));
```

Implementing t-digit chopping arithmetic in Maple is more difficult and requires a sequence of steps or a procedure. Exercise 12 explores this problem.

EXERCISE SET 1.3

1. Compute the absolute error and relative error in approximations of p by p^*.

   (a) $p = \pi$, $p^* = \frac{22}{7}$

   (b) $p = \pi$, $p^* = 3.1416$

   (c) $p = e$, $p^* = 2.718$

   (d) $p = \sqrt{2}$, $p^* = 1.414$

   (e) $p = e^{10}$, $p^* = 22000$

   (f) $p = 10^{\pi}$, $p^* = 1400$

   (g) $p = 8!$, $p^* = 39900$

   (h) $p = 9!$, $p^* = \sqrt{18\pi}\,(9/e)^9$

2. Perform the following computations (i) exactly, (ii) using three-digit chopping arithmetic, and (iii) using three-digit rounding arithmetic. (iv) Compute the relative errors in parts (ii) and (iii).

   (a) $\dfrac{4}{5} + \dfrac{1}{3}$

   (b) $\dfrac{4}{5} \cdot \dfrac{1}{3}$

   (c) $\left(\dfrac{1}{3} - \dfrac{3}{11}\right) + \dfrac{3}{20}$

   (d) $\left(\dfrac{1}{3} + \dfrac{3}{11}\right) - \dfrac{3}{20}$

3. Use three-digit rounding arithmetic to perform the following calculations. Compute the absolute error and relative error with the exact value determined to at least five digits.

   (a) $133 + 0.921$

   (b) $133 - 0.499$

   (c) $(121 - 0.327) - 119$

   (d) $(121 - 119) - 0.327$

   (e) $\dfrac{\frac{13}{14} - \frac{6}{7}}{2e - 5.4}$

   (f) $-10\pi + 6e - \dfrac{3}{62}$

   (g) $\left(\dfrac{2}{9}\right) \cdot \left(\dfrac{9}{7}\right)$

   (h) $\dfrac{\pi - \frac{22}{7}}{\frac{1}{17}}$

4. Repeat Exercise 3 using three-digit chopping arithmetic.

5. Repeat Exercise 3 using four-digit rounding arithmetic.

6. Repeat Exercise 3 using four-digit chopping arithmetic.

7. The first three nonzero terms of the Maclaurin series for the $\arctan x$ are $x - \frac{1}{3}x^3 + \frac{1}{5}x^5$. Compute the absolute error and relative error in the following approximations of π using the polynomial in place of the $\arctan x$:

   (a) $4\left[\arctan\left(\frac{1}{2}\right) + \arctan\left(\frac{1}{3}\right)\right]$ (b) $16\arctan\left(\frac{1}{5}\right) - 4\arctan\left(\frac{1}{239}\right)$

8. The two-by-two linear system

$$\begin{aligned} ax + by &= e, \\ cx + dy &= f, \end{aligned}$$

   where a, b, c, d, e, f are given, can be solved for x and y as follows:

$$\begin{aligned} \text{set } m &= \frac{c}{a}, \quad \text{provided } a \neq 0; \\ d_1 &= d - mb; \\ f_1 &= f - me; \\ y &= \frac{f_1}{d_1}; \\ x &= \frac{(e - by)}{a}. \end{aligned}$$

   Solve the following linear systems using four-digit rounding arithmetic.

   (a) $\begin{aligned} 1.130x - 6.990y &= 14.20 \\ 8.110x + 12.20y &= -0.1370 \end{aligned}$ (b) $\begin{aligned} 1.013x - 6.099y &= 14.22 \\ -18.11x + 112.2y &= -0.1376 \end{aligned}$

9. Suppose the points (x_0, y_0) and (x_1, y_1) are on a straight line with $y_1 \neq y_0$. Two formulas are available to find the x-intercept of the line:

$$x = \frac{x_0 y_1 - x_1 y_0}{y_1 - y_0} \quad \text{and} \quad x = x_0 - \frac{(x_1 - x_0)y_0}{y_1 - y_0}.$$

   (a) Show that both formulas are algebraically correct.

   (b) Use the data $(x_0, y_0) = (1.31, 3.24)$ and $(x_1, y_1) = (1.93, 4.76)$ and three-digit rounding arithmetic to compute the x-intercept both ways. Which method is better, and why?

10. The Taylor polynomial of degree n for $f(x) = e^x$ is $\sum_{i=0}^{n} x^i/i!$. Use the Taylor polynomial of degree nine and three-digit chopping arithmetic to find an approximation to e^{-5} by each of the following methods.

(a) $e^{-5} \approx \displaystyle\sum_{i=0}^{9} \frac{(-5)^i}{i!} = \sum_{i=0}^{9} \frac{(-1)^i 5^i}{i!}$

(b) $e^{-5} = \dfrac{1}{e^5} \approx \dfrac{1}{\sum_{i=0}^{9} 5^i/i!}$

An approximate value of e^{-5} correct to three digits is 6.74×10^{-3}. Which formula, (a) or (b), gives the most accuracy, and why?

11. A rectangular parallelepiped has sides 3 cm, 4 cm, and 5 cm, measured to the nearest centimeter.

    (a) What are the best upper and lower bounds for the volume of this parallelepiped?

    (b) What are the best upper and lower bounds for the surface area?

12. The following Maple procedure chops a floating-point number x to t digits.

```
chop:=proc(x,t);
 if x=0 then 0
 else
 e:=trunc(evalf(log10(abs(x))));
 if e>0 then e:=e+1 fi;
 x2:=evalf(trunc(x*10^(t-e))*10^(e-t));
 fi
end;
```

Verify that the procedure works for the following values.

(a) $x = 124.031$, $t = 5$

(b) $x = 124.036$, $t = 5$

(c) $x = -0.00653$, $t = 2$

(d) $x = -0.00656$, $t = 2$

1.4 Errors in Scientific Computation

In the previous section we saw how computational devices represent and manipulate numbers using finite-digit arithmetic. We now examine how the problems with this arithmetic can compound and look at ways to arrange arithmetic calculations to reduce this inaccuracy.

The loss of accuracy due to round-off error can often be avoided by a careful sequencing of operations or a reformulation of the problem. This is most easily described by considering a common computational problem.

EXAMPLE 1 The quadratic formula states that the roots of $ax^2 + bx + c = 0$, when $a \neq 0$, are

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \quad \text{and} \quad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}.$$

Consider this formula applied, using four-digit rounding arithmetic, to the equation $x^2 + 62.10x + 1 = 0$, whose roots are approximately $x_1 = -0.01610723$ and $x_2 = -62.08390$. In this equation, b^2 is much larger than $4ac$, so the numerator in the calculation for x_1 involves the *subtraction* of nearly equal numbers. Since

$$\sqrt{b^2 - 4ac} = \sqrt{(62.10)^2 - (4.000)(1.000)(1.000)} = \sqrt{3856 - 4.000} = 62.06,$$

we have

$$fl(x_1) = \frac{-62.10 + 62.06}{2.000} = \frac{-0.04000}{2.000} = -0.02000,$$

a poor approximation to $x_1 = -0.01611$ with the large relative error

$$\frac{|-0.01611 + 0.02000|}{|-0.01611|} = 2.4 \times 10^{-1}.$$

On the other hand, the calculation for x_2 involves the *addition* of the nearly equal numbers $-b$ and $-\sqrt{b^2 - 4ac}$. This presents no problem since

$$fl(x_2) = \frac{-62.10 - 62.06}{2.000} = \frac{-124.2}{2.000} = -62.10$$

has the small relative error

$$\frac{|-62.08 + 62.10|}{|-62.08|} = 3.2 \times 10^{-4}.$$

To obtain a more accurate four-digit rounding approximation for x_1, we can change the form of the quadratic formula by *rationalizing the numerator*:

$$x_1 = \left(\frac{-b + \sqrt{b^2 - 4ac}}{2a}\right)\left(\frac{-b - \sqrt{b^2 - 4ac}}{-b - \sqrt{b^2 - 4ac}}\right) = \frac{b^2 - (b^2 - 4ac)}{2a(-b - \sqrt{b^2 - 4ac}\,)},$$

which simplifies to

$$x_1 = \frac{-2c}{b + \sqrt{b^2 - 4ac}}.$$

Table 1.1:

	x	x^2	x^3	$6.1x^2$	$3.2x$
Exact	4.71	22.1841	104.487111	135.32301	15.072
Three-digit (chopping)	4.71	22.1	104.	134.	15.0
Three-digit (rounding)	4.71	22.2	105.	135.	15.1

Using this form of the equation gives

$$fl(x_1) = \frac{-2.000}{62.10 + 62.06} = \frac{-2.000}{124.2} = -0.01610,$$

which has the small relative error 6.2×10^{-4}. □

The rationalization technique in Example 1 can also be applied to give an alternative formula for x_2:

$$x_2 = \frac{-2c}{b - \sqrt{b^2 - 4ac}}.$$

This is the form to use if b is negative. In Example 1, however, the use of this formula results in the subtraction of nearly equal numbers, which produces the result

$$fl(x_2) = \frac{-2.000}{62.10 - 62.06} = \frac{-2.000}{0.04000} = -50.00,$$

with the large relative error 1.9×10^{-1}.

EXAMPLE 2 Evaluate $f(x) = x^3 - 6.1x^2 + 3.2x + 1.5$ at $x = 4.71$ using three-digit arithmetic.

Table 1.1 gives the intermediate results in the calculations. Carefully verify these results to be sure that your notion of finite-digit arithmetic is correct. Note that the three-digit chopping values simply retain the leading three digits, with no rounding involved, and differ significantly from the three-digit rounding values.

$$\begin{aligned}
\text{Exact:}\quad f(4.71) &= 104.487111 - 135.32301 + 15.072 + 1.5 \\
&= -14.263899; \\
\text{Three-digit (chopping):}\quad f(4.71) &= ((104. - 134.) + 15.0) + 1.5 = -13.5; \\
\text{Three-digit (rounding):}\quad f(4.71) &= ((105. - 135.) + 15.1) + 1.5 = -13.4.
\end{aligned}$$

The relative errors for the three-digit methods are

$$\left| \frac{-14.263899 + 13.5}{-14.263899} \right| \approx 0.05 \quad \text{for chopping}$$

and

$$\left|\frac{-14.263899+13.4}{-14.263899}\right| \approx 0.06 \quad \text{for rounding.}$$

As an alternative approach, $f(x)$ can be written in a **nested** manner as

$$f(x) = x^3 - 6.1x^2 + 3.2x + 1.5 = ((x-6.1)x + 3.2)x + 1.5.$$

This gives

$$\text{Three-digit (chopping):} \quad f(4.71) = ((4.71-6.1)4.71 + 3.2)4.71 + 1.5 = -14.2$$

and a three-digit rounding answer of -14.3. The new relative errors are

$$\begin{aligned} \text{Three-digit (chopping):} \quad & \left|\frac{-14.263899+14.2}{-14.263899}\right| && \approx && 0.0045; \\ \text{Three-digit (rounding):} \quad & \left|\frac{-14.263899+14.3}{-14.263899}\right| && \approx && 0.0025. \end{aligned}$$

Nesting has reduced the relative error for the chopping approximation to less than one-tenth that obtained initially. For the rounding approximation the improvement has been even more dramatic: the error has been reduced by more than 95%. Nested multiplication should be performed whenever a polynomial is evaluated since it minimizes the number of error producing computations. □

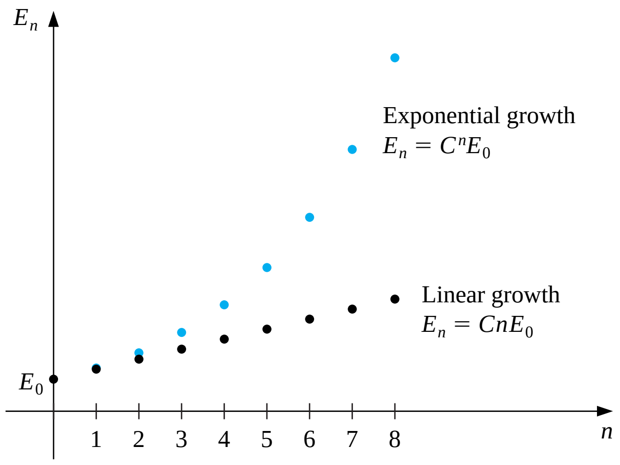
We will be considering a variety of approximation problems throughout the text, and in each case we need to determine approximation methods that produce dependably accurate results for a wide class of problems. Because of the differing ways in which the approximation methods are derived, we need a variety of conditions to categorize their accuracy. Not all of these conditions will be appropriate for any particular problem.

One criterion we will impose, whenever possible, is that of **stability**. A method is called **stable** if small changes in the initial data produce correspondingly small changes in the final results. When it is possible to have small changes in the initial date producing large changes in the final results, the method is **unstable**. Some methods are stable only for certain choices of initial data. These methods are called *conditionally stable*. We attempt to characterize stability properties whenever possible.

One of the most important topics effecting the stability of a method is the way in which the round-off error grows as the method is successively applied. Suppose an error with magnitude $E_0 > 0$ is introduced at some stage in the calculations and that the magnitude of the error after n subsequent operations is E_n. There are two distinct cases that often arise in practice. If a constant C exists independent of n, with $E_n \approx CnE_0$, the growth of error is **linear.** If a constant $C > 1$ exists independent of n, with $E_n \approx C^n E_0$, the growth of error is **exponential**. (It would be unlikely to have $E_n \approx C^n E_0$, with $C < 1$, since this implies that the error tends to zero.)

Linear growth of error is usually unavoidable and, when C and E_0 are small, the results are generally acceptable. Methods having exponential growth of error should be avoided, since the term C^n becomes large for even relatively small values of n and E_0. As a consequence, a method that exhibits linear error growth is stable, while one exhibiting exponential error growth is unstable. (See Figure 1.9 .)

Figure 1.9



Since iterative techniques involving sequences are often used, the section concludes with a brief discussion of some terminology used to describe the rate at which convergence occurs when employing a numerical technique. In general, we would like to choose techniques that converge as rapidly as possible. The following definition is used to compare the convergence rates of various methods.

Suppose that $\{\alpha_n\}_{n=1}^{\infty}$ is a sequence that converges to a number α as n becomes large. If positive constants p and K exist with

$$|\alpha - \alpha_n| \le \frac{K}{n^p}, \quad \text{for all large values of } n,$$

then we say that $\{\alpha_n\}$ **converges to α with rate, or order, of convergence $O(1/n^p)$** (read "big oh of $1/n^p$"). This is indicated by writing $\alpha_n = \alpha + O(1/n^p)$ and stated as "$\alpha_n \to \alpha$ with rate of convergence $1/n^p$." We are generally interested in the *largest* value of p for which $\alpha_n = \alpha + O(1/n^p)$.

We also use the "big oh" notation to describe how some divergent sequences grow as n becomes large. If positive constants p and K exist with

$$|\alpha_n| \le Kn^p, \quad \text{for all large values of } n,$$

Table 1.2:

n	1	2	3	4	5	6	7
α_n	2.00000	0.75000	0.44444	0.31250	0.24000	0.19444	0.16327
$\hat{\alpha}_n$	4.00000	0.62500	0.22222	0.10938	0.064000	0.041667	0.029155

then we say that $\{\alpha_n\}$ **goes to ∞ with rate, or order, $\mathbf{O(n^p)}$**. In the case of a sequence that becomes infinite, we are interested in the *smallest* value of p for which α_n is $O(n^p)$.

The "big oh" definition for sequences can be extended to incorporate more general sequences, but the definition as presented here is sufficient for our purposes.

EXAMPLE 3 Suppose that the sequences $\{\alpha_n\}$ and $\{\hat{\alpha}_n\}$ are described by

$$\alpha_n = \frac{n+1}{n^2} \qquad \text{and} \qquad \hat{\alpha}_n = \frac{n+3}{n^3}.$$

Although both $\lim_{n\to\infty} \alpha_n = 0$ and $\lim_{n\to\infty} \hat{\alpha}_n = 0$, the sequence $\{\hat{\alpha}_n\}$ converges to this limit much faster than does $\{\alpha_n\}$. This can be seen from the five-digit rounding entries for the sequences shown in Table 1.2.

Since

$$|\alpha_n - 0| = \frac{n+1}{n^2} \le \frac{n+n}{n^2} = 2 \cdot \frac{1}{n}$$

and

$$|\hat{\alpha}_n - 0| = \frac{n+3}{n^3} \le \frac{n+3n}{n^3} = 4 \cdot \frac{1}{n^2},$$

we have

$$\alpha_n = 0 + O\left(\frac{1}{n}\right) \quad \text{and} \quad \hat{\alpha}_n = 0 + O\left(\frac{1}{n^2}\right).$$

This result implies that the convergence of the sequence $\{\alpha_n\}$ is similar to the convergence of $\{1/n\}$ to zero. The sequence $\{\hat{\alpha}_n\}$ converges in a manner similar to the faster-converging sequence $\{1/n^2\}$. □

We also use the "big oh" concept to describe the rate of convergence of functions, particularly when the independent variable approaches zero.

Suppose that F is a function that converges to a number L as h goes to zero. If positive constants p and K exist with

$$|F(h) - L| \le Kh^p, \qquad \text{as } h \to 0,$$

then **$\mathbf{F(h)}$ converges to $\mathbf{L}$ with rate, or order, of convergence $\mathbf{O(h^p)}$**. This is written as $F(h) = L + O(h^p)$ and stated as "$F(h) \to L$ with rate of convergence h^p."

We are generally interested in the *largest* value of p for which $F(h) = L + O(h^p)$.

The "big oh" definition for functions can also be extended to incorporate more general zero-converging functions in place of h^p.

EXERCISE SET 1.4

1. (i) Use four-digit rounding arithmetic and the formulas of Example 1 to find the most accurate approximations to the roots of the following quadratic equations. (ii) Compute the absolute errors and relative errors for these approximations.

   (a) $\frac{1}{3}x^2 - \frac{123}{4}x + \frac{1}{6} = 0$ (b) $\frac{1}{3}x^2 + \frac{123}{4}x - \frac{1}{6} = 0$

   (c) $1.002x^2 - 11.01x + 0.01265 = 0$ (d) $1.002x^2 + 11.01x + 0.01265 = 0$

2. Repeat Exercise 1 using four-digit chopping arithmetic.

3. Let $f(x) = 1.013x^5 - 5.262x^3 - 0.01732x^2 + 0.8389x - 1.912$.

   (a) Evaluate $f(2.279)$ by first calculating $(2.279)^2$, $(2.279)^3$, $(2.279)^4$, and $(2.279)^5$ using four-digit rounding arithmetic.

   (b) Evaluate $f(2.279)$ using the formula

   $$f(x) = (((1.013x^2 - 5.262)x - 0.01732)x + 0.8389)x - 1.912$$

   and four-digit rounding arithmetic.

   (c) Compute the absolute and relative errors in parts (a) and (b).

4. Repeat Exercise 3 using four-digit chopping arithmetic.

5. The fifth Maclaurin polynomials for e^{2x} and e^{-2x} are

   $$P_5(x) = \left(\left(\left(\left(\frac{4}{15}x + \frac{2}{3}\right)x + \frac{4}{3}\right)x + 2\right)x + 2\right)x + 1$$

   and

   $$\hat{P}_5(x) = \left(\left(\left(\left(-\frac{4}{15}x + \frac{2}{3}\right)x - \frac{4}{3}\right)x + 2\right)x - 2\right)x + 1$$

   (a) Approximate $e^{-0.98}$ using $\hat{P}_5(0.49)$ and four-digit rounding arithmetic.

   (b) Compute the absolute and relative error for the approximation in part (a).

   (c) Approximate $e^{-0.98}$ using $1/P_5(0.49)$ and four-digit rounding arithmetic.

   (d) Compute the absolute and relative errors for the approximation in part (c).

6. (a) Show that the polynomial nesting technique described in Example 2 can also be applied to the evaluation of

$$f(x) = 1.01e^{4x} - 4.62e^{3x} - 3.11e^{2x} + 12.2e^{x} - 1.99.$$

   (b) Use three-digit rounding arithmetic, the assumption that $e^{1.53} = 4.62$, and the fact that $e^{n(1.53)} = (e^{1.53})^n$ to evaluate $f(1.53)$ as given in part (a).

   (c) Redo the calculation in part (b) by first nesting the calculations.

   (d) Compare the approximations in parts (b) and (c) to the true three-digit result $f(1.53) = -7.61$.

7. Use three-digit chopping arithmetic to compute the sum $\sum_{i=1}^{10} 1/i^2$ first by $\frac{1}{1} + \frac{1}{4} + \cdots + \frac{1}{100}$ and then by $\frac{1}{100} + \frac{1}{81} + \cdots + \frac{1}{1}$. Which method is more accurate, and why?

8. The Maclaurin series for the arctangent function converges for $-1 < x \leq 1$ and is given by

$$\arctan x = \lim_{n\to\infty} P_n(x) = \lim_{n\to\infty} \sum_{i=1}^{n} (-1)^{i+1} \frac{x^{2i-1}}{(2i-1)}.$$

   (a) Use the fact that $\tan \pi/4 = 1$ to determine the number of terms of the series that need to be summed to ensure that $|4P_n(1) - \pi| < 10^{-3}$.

   (b) The C programming language requires the value of π to be within 10^{-10}. How many terms of the series would we need to sum to obtain this degree of accuracy?

9. The number e is defined by $e = \sum_{n=0}^{\infty} 1/n!$, where $n! = n(n-1)\cdots 2 \cdot 1$, for $n \neq 0$ and $0! = 1$. (i) Use four-digit chopping arithmetic to compute the following approximations to e. (ii) Compute absolute and relative errors for these approximations.

   (a) $\displaystyle\sum_{n=0}^{5} \frac{1}{n!}$

   (b) $\displaystyle\sum_{j=0}^{5} \frac{1}{(5-j)!}$

   (c) $\displaystyle\sum_{n=0}^{10} \frac{1}{n!}$

   (d) $\displaystyle\sum_{j=0}^{10} \frac{1}{(10-j)!}$

10. Find the rates of convergence of the following sequences as $n \to \infty$.

    (a) $\lim_{n\to\infty} \sin\left(\frac{1}{n}\right) = 0$

    (b) $\lim_{n\to\infty} \sin\left(\frac{1}{n^2}\right) = 0$

    (c) $\lim_{n\to\infty} \left(\sin\left(\frac{1}{n}\right)\right)^2 = 0$

    (d) $\lim_{n\to\infty} [\ln(n+1) - \ln(n)] = 0$

11. Find the rates of convergence of the following functions as $h \to 0$.

    (a) $\lim_{h\to 0} \frac{\sin h - h\cos h}{h} = 0$

    (b) $\lim_{h\to 0} \frac{1-e^h}{h} = -1$

    (c) $\lim_{h\to 0} \frac{\sin h}{h} = 1$

    (d) $\lim_{h\to 0} \frac{1-\cos h}{h} = 0$

12. (a) How many multiplications and additions are required to determine a sum of the form
$$\sum_{i=1}^{n}\sum_{j=1}^{i} a_i b_j?$$

    (b) Modify the sum in part (a) to an equivalent form that reduces the number of computations.

13. The sequence $\{F_n\}$ described by $F_0 = 1, F_1 = 1$, and $F_{n+2} = F_n + F_{n+1}$, if $n \geq 0$, is called a *Fibonacci sequence*. Its terms occur naturally in many botanical species, particularly those with petals or scales arranged in the form of a logarithmic spiral. Consider the sequence $\{x_n\}$, where $x_n = F_{n+1}/F_n$. Assuming that $\lim_{n\to\infty} x_n = x$ exists, show that x is the *golden ratio* $(1+\sqrt{5})/2$.

14. The Fibonacci sequence also satisfies the equation
$$F_n \equiv \tilde{F}_n = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right].$$

    (a) Write a Maple procedure to calculate F_{100}.

    (b) Use Maple with the default value of `Digits` followed by `evalf` to calculate
$\tilde{F}_{100}$.

(c) Why is the result from part (a) more accurate than the result from part (b)?

(d) Why is the result from part (b) obtained more rapidly than the result from part (a)?

(e) What results when you use the command `simplify` instead of `evalf` to compute $\tilde{F}_{100}$?

15. The harmonic series $1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots$ diverges, but the sequence $\gamma_n = 1 + \frac{1}{2} + \cdots + \frac{1}{n} - \ln n$ converges, since $\{\gamma_n\}$ is a bounded, nonincreasing sequence. The limit $\gamma \approx 0.5772156649\ldots$ of the sequence $\{\gamma_n\}$ is called *Euler's constant.*

(a) Use the default value of `Digits` in Maple to determine the value of n for γ_n to be within 10^{-2} of γ.

(b) Use the default value of `Digits` in Maple to determine the value of n for γ_n to be within 10^{-3} of γ.

(c) What happens if you use the default value of `Digits` in Maple to determine the value of n for γ_n to be within 10^{-4} of γ?

1.5 Computer Software

Computer software packages for approximating the numerical solutions to problems are available in many forms. With this book, we have provided programs written in C, Fortran 77, Maple, Mathematica, MATLAB, and Pascal that can be used to solve the problems given in the examples and exercises. These programs will give satisfactory results for most problems that a student might need to solve, but they are what we call *special-purpose* programs. We use this term to distinguish these programs from those available in the standard mathematical subroutine libraries. The programs in these packages will be called *general purpose*.

The programs in general-purpose software packages differ in their intent from the programs provided with this book. General-purpose software packages consider ways to reduce errors due to machine rounding, underflow, and overflow. They also describe the range of input that will lead to results of a certain specified accuracy. Since these are machine-dependent characteristics, general-purpose software packages use parameters that describe the floating-point characteristics of the machine being used for computations.

There are many forms of general-purpose numerical software available commercially and in the public domain. Most of the early software was written for mainframe computers, and a good reference for this is *Sources and Development of Mathematical Software*, edited by Wayne Crowell [Cr]. Now that the desktop computer has become sufficiently powerful, standard numerical software is available for personal computers and workstations. Most of this numerical software is written in Fortran 77, although some packages are written in C, C++, and Fortran 90.

ALGOL procedures were presented for matrix computations in 1971 in [WR]. A package of FORTRAN subroutines based mainly on the ALGOL procedures was then developed into the EISPACK routines. These routines are documented in the manuals published by Springer-Verlag as part of their Lecture Notes in Computer Science series [SBIKM] and [GBDM]. The FORTRAN subroutines are used to compute eigenvalues and eigenvectors for a variety of different types of matrices. The EISPACK project was the first large-scale numerical software package to be made available in the public domain and led the way for many packages to follow. EISPACK is mantained by netlib and can be found on the Internet at http://www.netlib.org/eispack.

LINPACK is a package of Fortran 77 subroutines for analyzing and solving systems of linear equations and solving linear least squares problems. The documentation for this package is contained in [DBMS] and located on the Internet at http://www.netlib.org/linpack. A step-by-step introduction to LINPACK, EISPACK, and BLAS (Basic Linear Algebra Subprograms) is given in [CV].

The LAPACK package, first available in 1992, is a library of Fortran 77 subroutines that supersedes LINPACK and EISPACK by integrating these two sets of algorithms into a unified and updated package. The software has been restructured to achieve greater efficiency on vector processors and other high-performance or shared-memory multiprocessors. LAPACK is expanded in depth and breadth in version 3.0, which is available in Fortran 77, Fortran 90, C, C++, and JAVA. Fortran 90, C, and JAVA are only available as language interfaces or translations of the

FORTRAN libraries of LAPACK. The package BLAS is not a part of LAPACK, but the code for BLAS is distributed with LAPACK. The *LAPACK User's Guide, 3rd ed.* [An] is available from SIAM or on the Internet at
http://www.netlib.org/lapack/lug/lapack_lug.html.
The complete LAPACK or individual routines from LAPACK can be obtained through netlib at netlibornl.gov, netlibresearch.att.com, or http://www.netlib.org/lapack.

Other packages for solving specific types of problems are available in the public domain. Information about these programs can be obtained through electronic mail by sending the line "help" to one of the following Internet addresses: netlibresearch.att.com, netlibornl.gov, netlibnac.no, or netlibdraci.cs.uow.edu.au or to the uucp address uunet!research!netlib.

These software packages are highly efficient, accurate, and reliable. They are thoroughly tested, and documentation is readily available. Although the packages are portable, it is a good idea to investigate the machine dependence and read the documentation thoroughly. The programs test for almost all special contingencies that might result in error and failures. At the end of each chapter we will discuss some of the appropriate general-purpose packages.

Commercially available packages also represent the state of the art in numerical methods. Their contents are often based on the public-domain packages but include methods in libraries for almost every type of problem.

IMSL (International Mathematical and Statistical Libraries) consists of the libraries MATH, STAT, and SFUN for numerical mathematics, statistics, and special functions, respectively. These libraries contain more than 900 subroutines originally available in Fortran 77 and now available in C, Fortran 90, and JAVA. These subroutines solve the most common numerical analysis problems. In 1970 IMSL became the first large-scale scientific library for mainframes. Since that time, the libraries have been made available for computer systems ranging from supercomputers to personal computers. The libraries are available commercially from Visual Numerics, 9990 Richmond Ave S400, Houston, TX 77042-4548, with Internet address http://www.vni.com. The packages are delivered in compiled form with extensive documentation. There is an example program for each routine as well as background reference information. IMSL contains methods for linear systems, eigensystem analysis, interpolation and approximation, integration and differentiation, differential equations, transforms, nonlinear equations, optimization, and basic matrix/vector operations. The library also contains extensive statistical routines.

The Numerical Algorithms Group (NAG) has been in existence in the United Kingdom since 1970. NAG offers more than 1000 subroutines in a Fortran 77 library, about 400 subroutines in a C library, over 200 subroutines in a Fortran 90 library, and an MPI FORTRAN numerical library for parallel machines and clusters of workstations or personal computers. A subset of their Fortran 77 library (the NAG Foundation Library) is available for personal computers and workstations where work space is limited. The NAG C Library, the Fortran 90 library, and the MPI FORTRAN library offer many of the same routines as the FORTRAN Library. The NAG user's manual includes instructions and examples, along with sample output for each of the routines. A useful introduction to the NAG routines is [Ph]. The NAG library contains routines to perform most standard numerical analysis tasks in a manner similar to those in the IMSL. It also includes some statistical routines and a set of graphic routines. The library is commercially available from Numerical Algorithms Group, Inc., 1400 Opus Place, Suite 200, Downers Grove, IL 60515–5702, with Internet address http://www.nag.com.

The IMSL and NAG packages are designed for the mathematician, scientist, or engineer who wishes to call high-quality FORTRAN subroutines from within a program. The documentation available with the commercial packages illustrates the typical driver program required to use the library routines. The next three software packages are stand-alone environments. When activated, the user enters commands to cause the package to solve a problem. However, each package allows programming within the command language.

MATLAB is a matrix laboratory that was originally a FORTRAN program published by Cleve Moler [Mo]. The laboratory is based mainly on the EISPACK and LINPACK subroutines, although functions such as nonlinear systems, numerical integration, cubic splines, curve fitting, optimization, ordinary differential equations, and graphical tools have been incorporated. MATLAB is currently written in C and assembler, and the PC version of this package requires a numeric coprocessor. The basic structure is to perform matrix operations, such as finding the eigenvalues of a matrix entered from the command line or from an external file via function calls. This is a powerful self-contained system that is especially useful for instruction in an applied linear algebra course. MATLAB has been available since 1985 and can be purchased from The MathWorks Inc., Cochituate Place, 24 Prime Park Way, Natick, MA 01760. The electronic mail address of The Mathworks is infomathworks.com, and the Internet address is http://www.mathworks.com. MATLAB software is designed to run on many computers, including IBM PC compatibles, APPLE Macintosh, and SUN workstations. A student version of MATLAB does not require a coprocessor but will use one if it is available.

The second package is GAUSS, a mathematical and statistical system produced by Lee E. Ediefson and Samuel D. Jones in 1985. It is coded mainly in assembler and based primarily on EISPACK and LINPACK. As in the case of MATLAB, integration/differentiation, nonlinear systems, fast Fourier transforms, and graphics are available. GAUSS is oriented less toward instruction in linear algebra and more toward statistical analysis of data. This package also uses a numeric coprocessor if one is available. It can be purchased from Aptech Systems, Inc., 23804 S.E. Kent-Kangley Road, Maple Valley, WA 98038 (infoaptech.com).

The third package is Maple, a computer algebra system developed in 1980 by the Symbolic Computational Group at the University of Waterloo. The design for the original Maple system is presented in the paper by B.W. Char, K.O. Geddes, W.M. Gentlemen, and G.H. Gonnet [CGGG]. Maple has been available since 1985 and can be purchased from Waterloo Maple Inc., 57 Erb Street, Waterloo, ON N2L 6C2. The electronic mail address of Waterloo Maple is infomaplesoft.com, and the Internet address is http://www.maplesoft.com. Maple, which is written in C, has the ability to manipulate information in a symbolic manner. This symbolic manipulation allows the user to obtain exact answers instead of numerical values. Maple can give exact answers to mathematical problems such as integrals, differential equations, and linear systems. Maple has the additional property of allowing worksheets, which contain written text and Maple commands. These worksheets can then be loaded into Maple and the commands executed. Because of the properties of symbolic computation, numerical computation, and worksheets, Maple is the language of choice for this text. Throughout the book Maple commands will be embedded into the text.

Numerous packages are available that can be classified as supercalculator packages for the PC. These should not be confused, however, with the general-purpose software listed here. If you have an interest in one of these packages, you should read *Supercalculators on the PC* by B. Simon and R. M. Wilson [SW].

Additional information about software and software libraries can be found in the books by Cody and Waite [CW] and by Kockler [Ko], and in the 1995 article by Dongarra and Walker [DW]. More information about floating-point computation can be found in the book by Chaitini-Chatelin and Frayse [CF] and the article by Goldberg [Go].

Books that address the application of numerical techniques on parallel computers include those by Schendell [Sche], Phillips and Freeman [PF], and Golub and Ortega [GO].